|  |  |
| --- | --- |
| **COURSEWORK** | |
| **Module Title: Programming** | **Module Code: 4COM1037 & 4COM1045** |
| **Assignment Title: Basic Game Development in Python** | **Semester: B, Individual Assignment** |
| **Tutor**: Dr Barry Ip, Dr Carolyn Devereux, Dr Kheng-Lee Koay, Dr Luke Wood, Dr Roberto Pascalis | **Internal Moderator**: Dr Peter Lane |

|  |  |
| --- | --- |
| Student ID Number **ONLY**: | Year Code: |
|  |  |

|  |  |
| --- | --- |
| Marks Awarded %: | Marks Awarded after Lateness Penalty applied %: |
| Penalties for Late Submissions   * Late submission of any item of coursework for each day or part thereof (or for hard copy submission only, working day or part thereof) for up to five days after the published deadline, coursework relating to modules at Levels 0, 4, 5, 6 submitted late (including deferred coursework, but with the exception of referred coursework), will have the numeric grade reduced by 10 grade points until or unless the numeric grade reaches or is 40. Where the numeric grade awarded for the assessment is less than 40, no lateness penalty will be applied. * Late submission of referred coursework will automatically be awarded a grade of zero (0). * Coursework (including deferred coursework) submitted later than five days (five working days in the case of hard copy submission) after the published deadline will be awarded a grade of zero (0). * Where genuine serious adverse circumstances apply, you may apply for an extension to the hand-in date, provided the extension is requested a reasonable period in advance of the deadline. | |
| Please refer to your student handbook for details about the grading schemes used by the School when assessing your work. Guidance on assessment will also be given in the Module Guide. | |
| Guidance on avoiding academic assessment offences such as plagiarism and collusion is given at this URL: <http://www.studynet.herts.ac.uk/ptl/common/LIS.nsf/lis/citing_menu> | |

|  |  |  |
| --- | --- | --- |
| **ASSIGNMENT BRIEF** | | |
| **This Assignment assesses the following module Learning Outcomes:**   * LO 1: Have knowledge and understanding of sufficient features of a high level programming language to develop solutions to simple programming problems; * LO 2: Have knowledge and understanding of the concepts of data declaration and operations, control flow (sequence, selection, iteration, subroutine call) and modularisation; * LO 3: Have knowledge and understanding of the terminology used in describing programs and programming. * LO 4: Design and implement solutions to simple programming problems in a given programming language; * LO 5: Execute, test and de-bug programs; * LO 6: Document programs to an agreed standard. | | |
| **Assignment Brief:**  Please see full details as set out below. | | |
| **Submission Requirements:**  Please note the following essential points:   * All submissions need to be made on Canvas, via the Assignments link on the module page. You should submit all program files as a single zipped file. * **No** additional modules should be included in your code other than those covered in this module (e.g. turtle, random, constants, and other .py files that you create for the project). This is to ensure markers can run your program with minimal errors and compatibility issues. * Your program should be written in Python 3. | | |
| This assignment is worth 60**%** of the overall assessment for this module.  **Marks awarded for:**   |  |  | | --- | --- | | **Components** | **Marks** | | Programming Technique   |  |  | | --- | --- | | *Appropriate data types and declarations* | *10%* | | *Appropriate program control techniques* | *10%* | | *Modular programming (functions, separate files, etc.)* | *10%* | | *Attempt of stretch features, if any* | *10%* | | *Appropriate comments on the program* | *10%* | | 50% | | Program Execution   |  |  | | --- | --- | | *Suitable input and output* | *5%* | | *Appropriate drawing and labelling of the game board on game screen* | *5%* | | *Correct positioning of game items on game screen (snakes, ladders, players, etc.)* | *5%* | | *Correct display of player prompts on screen (roll dice, dice number, new game, etc.)* | *5%* | | *Correct display of ‘Win’ message on game screen* | *5%* | | *Correct movement of players on game screen (according to dice roll, and landing on snakes and ladders)* | *10%* | | *Correct game restart on end of each instance* | *5%* | | *Successful execution of stretch features, if any* | *10%* | | 50% | | **Total** | **100%** |   Also see Page 7 for general grading criteria.  Please note:   1. For undergraduate modules, a score above 40% represent pass performance at honours level. 2. For postgraduate modules, a score of 50% or above represents a pass mark. 3. Modules may have several components of assessment and may require a pass in all elements. For further details, please consult the relevant Module Guide or ask the Module Leader. | | |
| Typical (hours) required by the student(s) to complete the assignment: **80** hours | | |
| **Date Work handed out:**  Week commencing 8th March 2021 | **Date Work to be handed in:**  2pm BST, Tuesday, 20th April 2021 | **Target Date for the return of the marked assignment:**  Week commencing 10th May 2021 |
| **Type of Feedback to be given for this assignment:**  Individual feedback will be given to each student. | | |

**Basic Game Development in Python**

**Demo Video:**

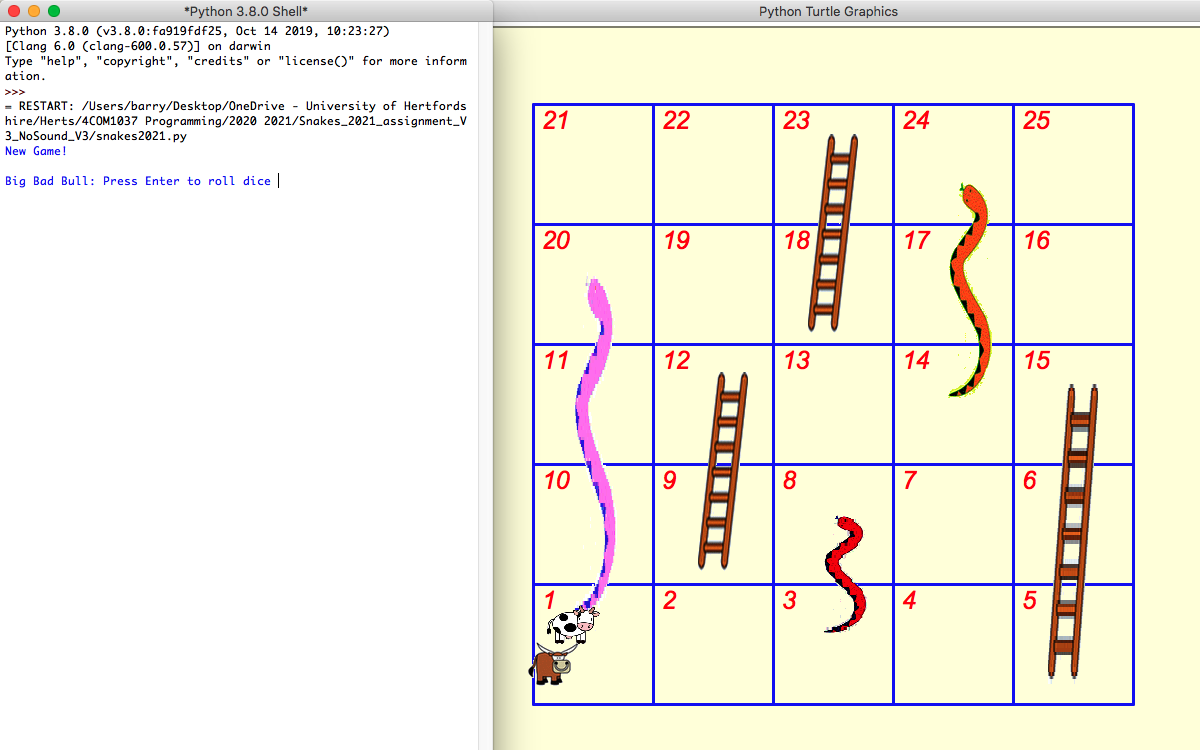
A showcase demonstration video (Coursework\_Demo.mov) can be found on Canvas, which can be used as guidance for the development of this coursework.

**Requirements:**

For this assignment, you are tasked to develop a Snakes and Ladders game that allows two players to participate in turn. The game should contain the following key elements of play:

1. The program should commence by drawing a suitable Snakes and Ladders board on the screen. The requirement is a 5 x 5 board, giving a total of 25 squares. Suitable labels should be printed on the board to indicate the numbers of the squares – see screenshot below for an example.
2. The board should contain three ladders and three snakes – see screenshot below for their required positions. (**Use the default images provided for the snakes and ladders as part of the assignment package.**)
3. Player 1 is represented by an image of a bull and Player 2 by an image of a cow (**also use the default images provided**). After the board is drawn, the bull and the cow should be positioned on square 1 for the start of a new game.
4. The game should start by prompting Player 1 to press a button to start the roll of the dice.
5. The program should generate a random number between 1 and 6 to indicate the roll of the dice. This number should either be printed to screen or shown as an image (**use images provided**).
6. Player 1 then moves the relevant number of spaces on the board: the program should make the necessary movement on behalf of the player, so no input is required – a printed confirmation of the player’s new location on the board may also be provided (e.g. ‘Player 1 is now on square 10’).
7. The above process is repeated for Player 2, and the cycle repeats until a player first reaches the final square with the required number of moves.
8. If a player lands on a snake or a ladder during their move, they **must** reposition to the correct location on the board (again, the relevant movement should be handled automatically by the program).
9. A suitable ‘winning’ image or prompt should be provided when a player wins (**use default image provided**).
10. The game should then repeat and the board reset ready for another session. Provide a printed confirmation of the start of a new game (see screenshot below). The game ends whenever a player closes the window.
11. The game should provide a suitable graphical interface to enhance the user’s experience and increase the game’s appeal (e.g. clear prompts, choice of colours, etc.).

The layout of the game screen should look something like this:



Take note of some important details:

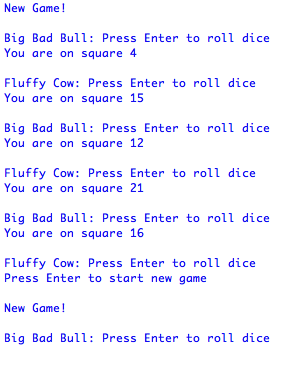
* There are three ladders with the following locations:
  + The first ladder starts on square 5 and leads to square 15
  + The second ladder starts on square 9 and leads to square 12
  + The third ladder starts on square 18 and leads to square 23
* There are three snakes with the following locations:
  + The first snake starts on square 8 and leads to square 3
  + The second snake starts on square 20 and leads to square 1
  + The third snake starts on square 24 and leads to square 14

You are **not** expected to change the location of the above.

User input (e.g. “press a button to roll the dice”) may take place in the Python console. The game should repeat automatically on the completion of each instance, i.e. when a player wins and a suitable message has been shown – see example below. Again, you may use the default image provided for the winning screen.



Example of messages and indication of a new game:



**Design Strategy:**

There are numerous approaches to implement this program. Here is one possible pathway to cover the essential points of development:

1. Ensure you understand the requirements of the design specification and try to document key areas by using pseudo code and flowcharts

(Refer to Lecture 1 on software design basics and associated tutorials)

1. Write a simple main file. Use turtle to draw a suitable board to screen and print the numbers for individual squares

(Refer to Lecture 9 for Python graphics, Lecture 4 for program structure, Lecture 6 on functions, and associated tutorials)

1. Position the relevant snakes, ladders, bull, and cow onto their required starting positions.

(Refer to Lecture 9 for Python graphics and associated tutorials)

1. Display relevant information in a user-friendly format: e.g. allow Player 1 and Player 2 to commence the game by rolling a dice in turn and show the rolled number on screen (Refer to Lecture 5 for I/O, Lecture 8 for program control, Lecture 12 for numbers and arithmetic, and associated tutorials)
2. Move the player to the relevant space according to the dice number

(See Lecture 9 for Python graphics, Lecture 8 for program control, and associated tutorials)

1. If a player lands on either a ladder or snake, move them to the correct location

(Refer to Lecture 9 for Python graphics, Lecture 8 for program control, and associated tutorials)

1. Determine if a player has reached the end square, and if so, display a win message

(Refer to Lecture 9 for Python graphics, Lecture 8 for program control, and associated tutorials)

1. Allow the game to repeat and reset the board

(Refer to Lecture 9 for Python graphics, Lecture 8 for program control, and associated tutorials)

1. Enhance the structure and efficiency of the program

(Refer to Lectures 5 and 6 for the creation of additional modules and functions, Lecture 7 for global and local variables, and associated tutorials)

1. Enhance the documentation of the program

(Refer to Lecture 4 for information on adding comments to the code)

1. Problem solve and debug as a natural course of development (try to be independent – tutors may advise, but cannot solve all the problems that may exist)

(Refer to Lecture 13 for debugging and problem solving)

**Stretch Abilities:**

If you wish to extend this project further, you may consider adding relevant features as you deem appropriate. Some examples include:

* Ensuring that a player rolls the correct number before they can win: for example, if a player is on square 23, they MUST roll 2 to win. If they roll more than 2, the program will make the necessary reverse steps (e.g. if currently on square 23, then roll a 6, player should be moved back to square 21).
* keeping track of score (i.e. the number of games won or lost by each player);
* allowing for player information to be entered and displayed (name, age, etc.);
* improving the structure of the code for efficiency and reusability (e.g. considered use of functions, separate files/modules, etc.)
* some other feature(s) you feel may be appropriate for a game of this type.

You can be as creative as you wish in order to enhance the quality and appeal of the game. However, I suggest you attempt stretch features only when **all** of the core features are complete and working correctly. Do not be overly ambitious: a partially working game is better than one that doesn’t work at all.

For any standard or additional features, please **do not** use any other libraries other than those covered in this module (e.g. random, constants, other function files that you write). This is to ensure that the tutors can run your programs with minimal errors and compatibility issues.

You do not need to use any other images or files beyond those provided as part of this assignment. However, if you do, please ensure you include all relevant images and program files in your submission as a single Zip file. Without these, the markers will **not** be able to run your program.

**Grading Criteria**

|  |  |  |
| --- | --- | --- |
| **Score** | **Programming Technique (50%)** | **Program Execution (50%)** |
| 95 – 100 (outstanding) | Solutions are novel and/or truly innovative & fully addresses task within context. Profound depth of engagement and incisive ideas development. Decision making is in perceptive; techniques are used in an outstanding manner. | Outstanding use of appropriate technology as applied to the problem domain. Consistently accurate and outstanding application of skills and techniques. Components are fully functional as required and consistently step beyond expectations using sophisticated solutions. |
| 85 – 94 (Excellent) | Solutions are novel & fully addresses task within context. Excellent depth of engagement and incisive ideas development. Decision making and techniques used in an exemplary manner. | Excellent use of appropriate technology as applied to the problem domain. Consistently accurate and exemplary application of skills and techniques. Components are fully functional as required and occasionally stepping beyond expectations using sophisticated solutions. |
| 70 – 79 (Very good) | Solutions relate directly to task and may step beyond conventions. Strong engagement with subject material and processes, evaluation of alternatives, solutions come from process. Strong use of techniques to derive solutions. | Very good use of appropriate technology as applied to the problem domain. Understanding is demonstrated. Small errors in technique and/or application with little impact on deliverables. Most required components are functional, with a high level and accurate application of skills and techniques. |
| 60 – 69  (Good) | Solutions are appropriate to task, work well within conventions. Good use of subject material and processes. Experimentation is in evidence to support implementation within conventions. Good use of techniques to derive solutions. | Good use of appropriate technology as applied to the problem domain. Good and reasonably accurate application of skills and techniques demonstrated. Some errors in technique and/or application with minor impact on deliverables; required components may not be fully functional but may be easily resolved. |
| 50 – 59  (Clear pass) | Solutions limited to task but address conventions. Solutions found or adopted but completion may be rushed. Some experiments but limited alternatives. Techniques are applied to derive solutions but steps are missed. | Satisfactory use of appropriate technology as applied to the problem domain. Satisfactory application of skills and techniques demonstrated but with minor inaccuracies. Some errors in technique and/or application with some impact on deliverables; required components may suffer from some errors that affect overall functionality, but which can be rectified by minor improvement. |
| 40 – 49 (Marginal pass) | Solutions quite limited to task and may only partially address conventions. Solutions found or adopted but completion may be rushed or incomplete. Some experiments but quite limited alternatives. Techniques are applied to derive solutions but key steps may be missed. | Some correct use of appropriate technology as applied to the problem domain. Some appropriate application of skills and techniques demonstrated but with inaccuracies. Some errors in technique and/or application with some impact on deliverables; required components may suffer from errors that affect overall functionality, and which need to be rectified by greater improvement. |
| 30 – 39 (Marginal fail) | Solutions frame task inappropriately and do not address conventions. Basic use of strategies, few alternatives, limited evaluation with limited experimentation. Limited use of techniques to derive solutions. | Limited use of appropriate technology as applied to the problem domain. Limited application of skills and techniques demonstrated. Many errors in technique and/or application with high impact on deliverables; required components may suffer from critical errors, and may require significant improvement. |
| 20 – 29  (Clear fail) | Lacking in appropriate solutions with very limited use of strategies, no evaluation and little evidence of ideas development. Little use of techniques to derive solutions. Work is below the minimum standard. | Very little use of appropriate technology as applied to the problem domain. Very little skill and application of techniques demonstrated. High number of errors with very high impact on deliverables; required components may not function in any discernable way and/or are entirely missing. Certain elements of academic misconduct. |
| 0 – 19  (Fail – nothing of merit) | No or completely inappropriate solution. No use of strategies, no planning and no experimentation. No application of techniques to derive solutions. Academic misconduct. | No use of appropriate technology as applied to the problem domain. No skill and application of technique demonstrated. Very high number of errors in deliverable or no deliverable submitted. Required components are under developed, missing, or inappropriate. Academic misconduct. |